



**RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE
SCIENTIFIQUE**

**UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE HOUARI
BOUMEDIENE**

Rapport TP3:

Etudiante1 : Hemaizi Siryne Zineb

Matricule : 222231516216

Section : ACAD A

Interruptions et programmation de quantum detemps à l'aide de l'interruption périodique 1CH

Partie I : afficherVingsVecteurs

```
data segment
N db ? ; var qui va contenir le num du
vecteur suivant
tableau db '0123456789ABCDEF' ; table de
correspondance

espace db " : $" ; pour separer le ip
du cs lors de l affichage
saut db 10, 13, '$'
msg db "Les vecteurs sont afficher $"
data ends

mapile segment stack
dw 128 dup(?)
tos label word
mapile ends

code segment
    Assume cs:code, ds:data, ss:mapile
    effaceEcran proc
        mov ax, 3
        int 10h
        ret
    effaceEcran endp

    setCursorPosition proc

        mov ah,2
        mov bh, 0
        inc dh          ; num de ligne
        mov dl, 30      ; num de colonne
        int 10h

        ret
    setCursorPosition endp

    getCursorPosition proc
        push cx
        mov ah, 3h
        int 10h
        pop cx ; dx contient les num de
lign et col courantes
        ret
    getCursorPosition endp

    afficherChaine proc
        push bp
        mov bp, sp
        mov dx, [bp+4]
        mov ah, 9h
        int 21h
        pop bp
        ret
    afficherChaine endp
```

```
afficherCaractereHexa proc near
    push bp
    mov bp, sp
    mov ax, [bp+4]
    lea bx, tableau ; table qui
contient les codes ascii des
caracteres hexa 0 a F
    xlat
    mov ah, 0eh
    int 10h

    pop bp
    ret 2
afficherCaractereHexa endp

; cette proc affiche un vecteur (ip-
cs) en exa ;

    afficherAdresseRoutineN proc
        push bp
        mov bp, sp
        xor ax, ax
        mov dx, [bp+4]
        ;valeur des 4 bits de poids fort
de l'octet fort de l'adresse
        mov al, dh
        shr al, 4
        push ax
        call afficherCaractereHexa
        ;valeur des 4 bits de poids faible
de de l'octet fort l'adresse
        mov al, dh
        and al, 0fh
        push ax
        call afficherCaractereHexa

        ;valeur des 4 bits de poids fort
de de l'octet faible de l'adresse
        mov al, dl
        shr al, 4
        push ax
        call afficherCaractereHexa

        ;valeur des 4 bits de poids faible
de de l'octet faible de l'adresse

        mov al, dl
        and al, 0fh
        push ax
        call afficherCaractereHexa

        pop bp
        ret 2
    afficherAdresseRoutineN endp
```

```

; cette proc affiche les 20 prochains vect d'its ;
afficherVingsVecteurs proc
    mov cx, 20
    vecSuivant:

    call setCursorPosition

    mov al, N ; N contient le numero du vecteur suivant a afficher
    mov ah, 35h
    int 21h
    push bx ; CONTIENT IP
    call afficherAdresseRoutineN

    mov dx, offset espace
    push dx
    call afficherChaine

    push es ; CONTIENT CS
    call afficherAdresseRoutineN

    inc N
    call getCursorPosition

    loop vecSuivant
    ret
afficherVingsVecteurs endp

; Programme appelant ;

start:mov ax, data
    mov ds, ax
    mov ax, mapile
    mov ss, ax
    lea sp, tos
    mov N, 0
encore:
    call afficherVingsVecteurs

    mov dx, offset saut
    push dx
    call afficherChaine

    mov dx, offset msg
    push dx
    call afficherChaine

    mov ah, 1h
    int 21h

    cmp al, 'o'
    jnz fin

    call effaceEcran

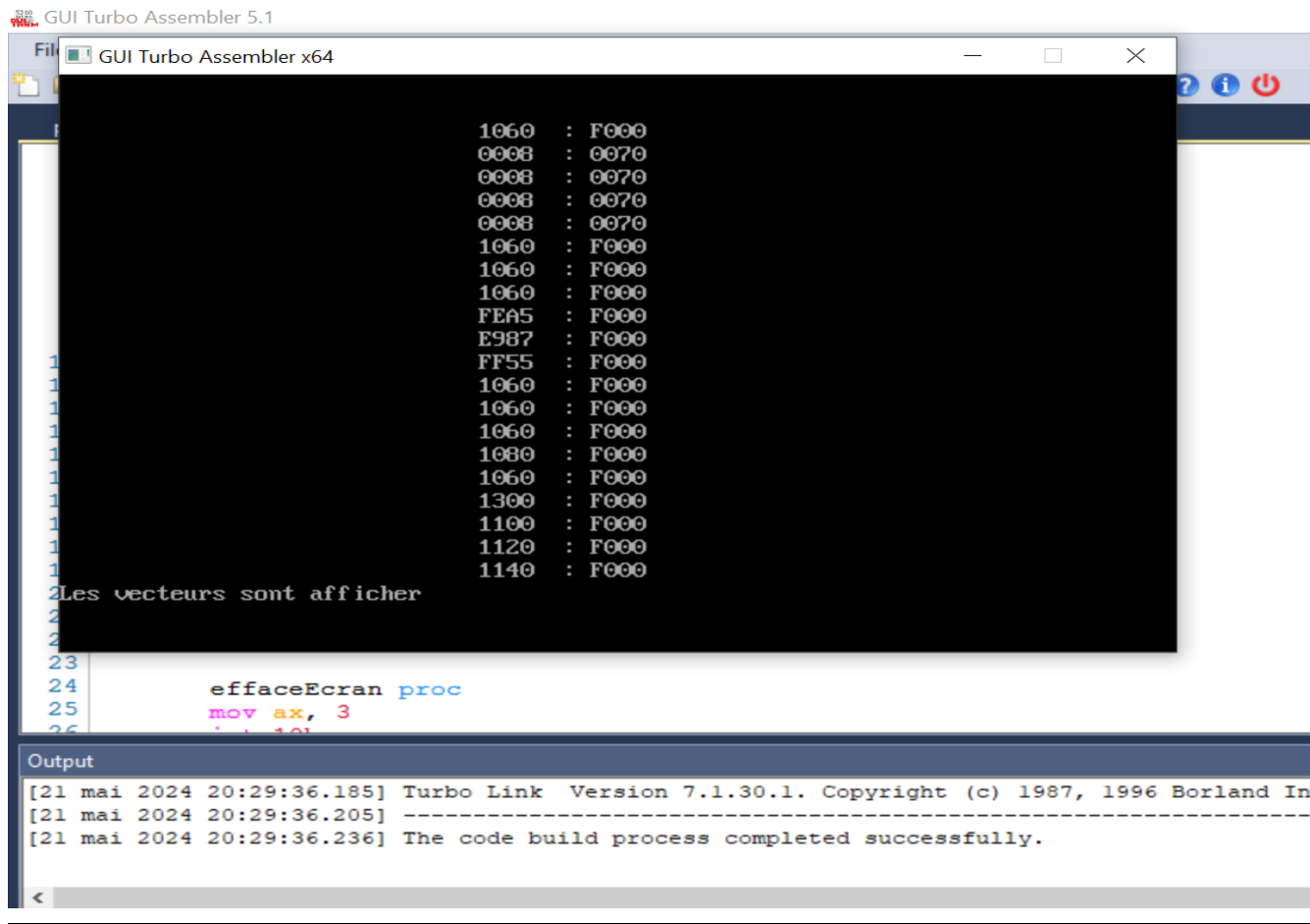
    jmp encore

fin:
    mov ah, 4ch
    int 21h

code ends
end start

```

Capture :



The screenshot shows the GUI Turbo Assembler 5.1 interface. The main window displays assembly code with addresses and hex values. Below the code, there is a section labeled 'Output' showing the build process results.

```
1060 : F000
0008 : 0070
0008 : 0070
0008 : 0070
0008 : 0070
1060 : F000
1060 : F000
1060 : F000
FEA5 : F000
E987 : F000
FF55 : F000
1060 : F000
1060 : F000
1060 : F000
1080 : F000
1060 : F000
1300 : F000
1100 : F000
1120 : F000
1140 : F000

2 Les vecteurs sont afficher

23
24 effaceEcran proc
25 mov ax, 3
26
```

Output

```
[21 mai 2024 20:29:36.185] Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland In
[21 mai 2024 20:29:36.205] -----
[21 mai 2024 20:29:36.236] The code build process completed successfully.
```

Partie II :

L'interruption 1CH, est une interruption logicielle fréquemment utilisée dans les systèmes d'exploitation pour gerer le temps implémentée pour :

- 1. Déclenchement périodique** : L'interruption 1CH est déclenchée périodiquement par un composant matériel appelé "timer" ou "horloge système". Dans les micro-ordinateurs, ce timer génère une interruption toutes les 18,2 fois par seconde, ce qui correspond à une périodicité d'environ 55 millisecondes. Ce déclenchement régulier permet au système d'exploitation de maintenir une mesure précise du temps.
- 2. Routine de gestion du timer** : Lorsque l'interruption 1CH est déclenchée, le processeur exécute automatiquement une routine de gestion du timer. Cette routine est généralement une fonction intégrée au système d'exploitation et est responsable de plusieurs tâches, telles que la mise à jour de l'horloge système, la gestion des temporisations dans les programmes en cours d'exécution, la planification des tâches, etc.
- 3. Utilisation dans les programmes** : Les programmes peuvent utiliser l'interruption 1CH pour implémenter des fonctionnalités liées au temps, telles que des temporisations précises, des animations, des horloges logicielles, etc. Ils peuvent installer leurs propres routines de gestion du timer pour répondre aux déclenchements de l'interruption 1CH et exécuter des actions spécifiques en conséquence.
- 4. Déroulage de l'interruption 1CH*** : Dans certains cas, les programmes peuvent dérouter l'interruption 1CH pour remplacer la routine de gestion du timer par leur propre code. Cela leur permet d'avoir un contrôle plus fin sur la gestion du temps et de personnaliser le comportement du système en fonction de leurs besoins spécifiques.

D'où , l'interruption 1CH est un mécanisme crucial pour la gestion du temps dans les systèmes d'exploitation. Son implémentation permet aux programmes de suivre le temps avec précision et de prendre des actions en fonction des intervalles de temps réguliers définis par le timer matériel.

Programme 1 :

```
data segment
    message db 10,13,"**** Programme principal
en cours **** $"
    deroute_msg db 10,13,"deroutement fait...
$"
    debut_msg db 10,13,"****Debut du quantum
de Temps Logiciel****",13,10,"$"
    sec_msg db "1 sec ecoulee..... $"
data ends

my_stack segment stack 'stack'
    dw 128 dup(?)
    TOP label word
my_stack ends

code segment
    assume cs:code, ds:data, ss:my_stack

output proc near
    mov ah,09h
    int 21h
    ret
output endp

; Variables pour le comptage du temps
time_counter dw 0
deroute_done db 0

deroute proc near
    push ax
    push dx
    push ds

    ; Si le deroutement a d?j? ?t?
    affich? , sauter l'affichage
    cmp deroute_done, 1
    je skip_display

    ; Afficher le message de deroutement
    mov ax, data
    mov ds, ax
    mov dx, offset deroute_msg
    call output
    ; Marquer le deroutement comme d?j?
    affich?
    mov deroute_done, 1

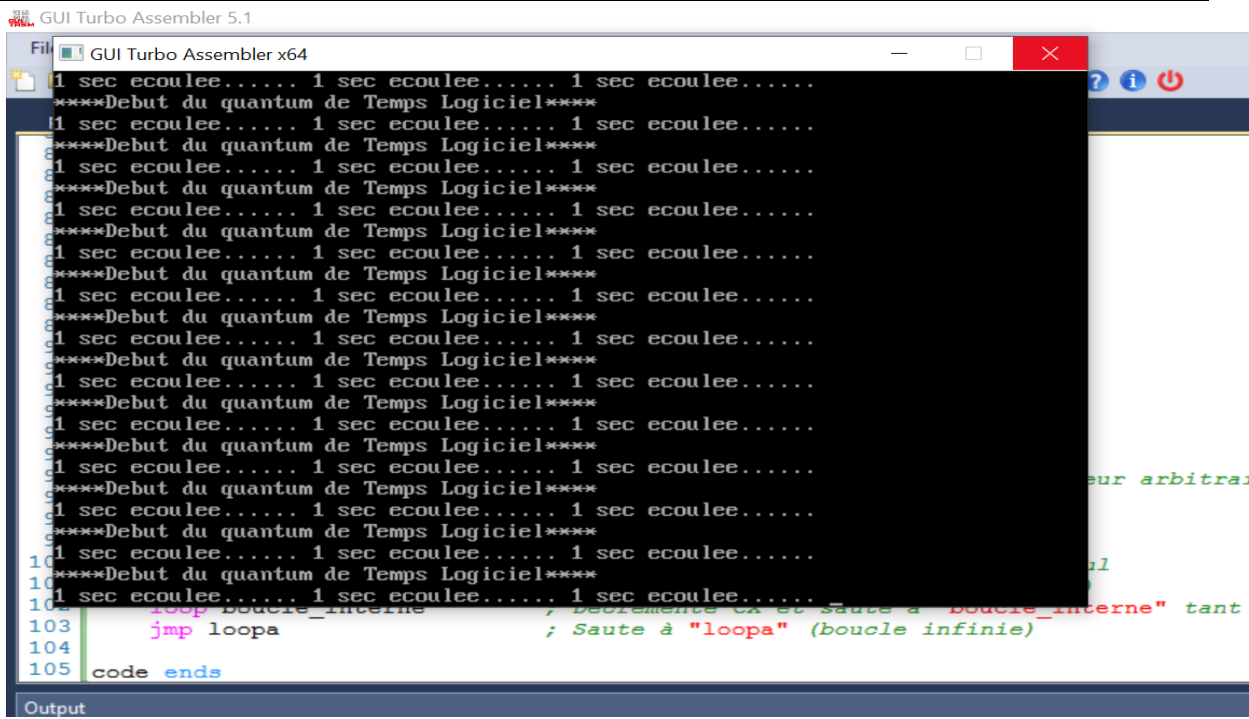
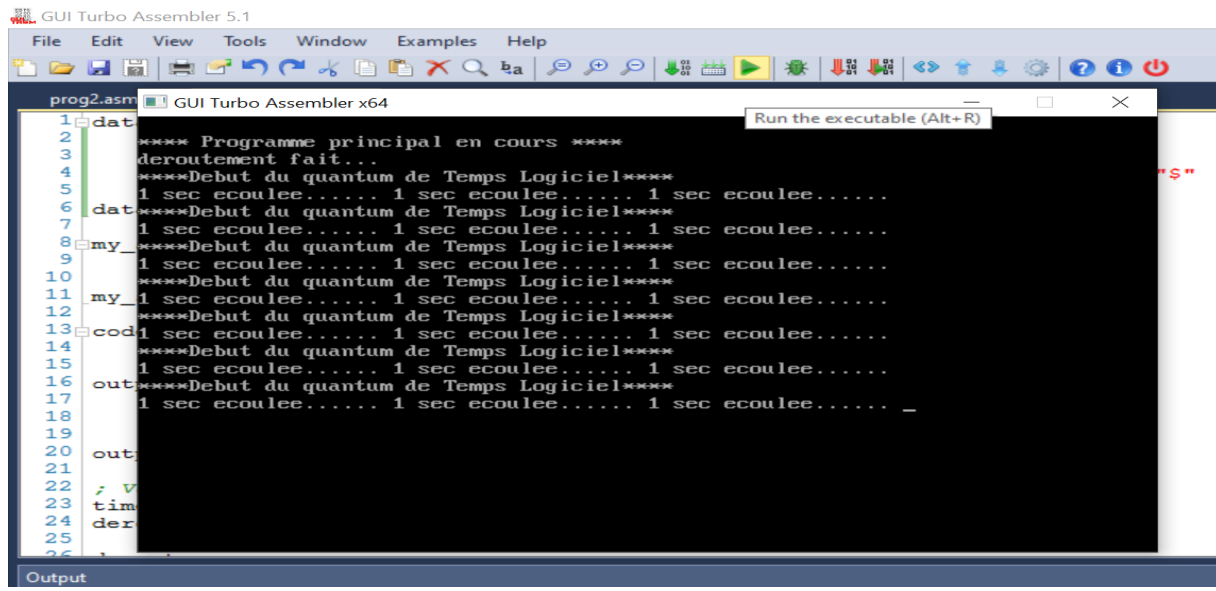
skip_display:
    ; Incr?menter le compteur de temps
    (18,2 fois par seconde)
    inc time_counter
    cmp time_counter, 18 ; Environ une
seconde (18 ticks)
    jb short skip_display
```

```
; R?initialiser le compteur de
temps
    mov time_counter, 0
    ; Afficher les messages
    mov ax, data
    mov ds, ax
    mov dx, offset debut_msg
    call output
    mov cx, 3 ; Boucle pour
afficher le message "1 sec
ecoulee....." trois fois
display_loop:
    mov dx, offset sec_msg
    call output
    loop display_loop

; Restaurer les registres
et revenir de l'interruption
    pop ds
    pop dx
    pop ax
    iret

deroute endp
installation proc near
    push ds
    mov ax,cs
    mov ds,ax
    mov dx,offset deroute
    mov ax,251Ch
    int 21h
    pop ds
    ret
installation endp
start:
    mov ax, data
    mov ds, ax
    mov ax, my_stack
    mov ss, ax
    mov sp, TOP
    call installation
    mov ax, 3
    int 10h ; Effacer l'ecran
en utilisant l'interruption 10h
    mov ax, 3
loopa:
    mov dx, offset message
    call output
mov cx, 3C0h
boucle_interne:
    inc bl
    mov ax, 3d09h
attente:
    dec ax
    jnz attente
    loop boucle_interne
    jmp loopa
end start
```

Capture :



Le programme affiche chaque second et arrete pas

Programme2 :

```
data segment
    PROG1 db "Tache 1 en cours d'execution...",10,13,"$" ; Message pour la tâche 1
    PROG2 db "Tache 2 en cours d'execution...",10,13,"$" ; Message pour la tâche 2
    PROG3 db "Tache 3 en cours d'execution...",10,13,"$" ; Message pour la tâche 3
    PROG4 db "Tache 4 en cours d'execution...",10,13,"$" ; Message pour la tâche 4
    PROG5 db "Tache 5 en cours d'execution...",10,13,"$" ; Message pour la tâche 5
    NEWLINE db 10,13,"$" ; Chaîne pour un saut de ligne
    INSTALLED db "Deroutement fait",10,13,"$" ; Message d'installation
    timer_counter dw 0 ; Compteur de temps pour compter 5 secondes
    current_task db 0 ; Indicateur de la tâche actuelle
data ends

code segment
    assume cs: code, ds: data

; Procédure d'installation de la routine d'interruption
installation proc near
    push ds
    mov ax,cs
    mov ds,ax
    mov dx,offset deroute
    mov ax,251CH
    int 21H
    pop ds
    mov dx,offset INSTALLED
    mov ah,09h
    int 21h
    ret
installation endp

; Proédures pour afficher les messages de chaque tâche
PRO1 proc near
    mov dx,offset PROG1
    mov ah,09h
    int 21h
    ret
PRO1 endp

PRO2 proc near
    mov dx,offset PROG2
    mov ah,09h
    int 21h
    ret
PRO2 endp

PRO3 proc near
    mov dx,offset PROG3
    mov ah,09h
    int 21h
    ret
PRO3 endp

PRO4 proc near
    mov dx,offset PROG4
    mov ah,09h
    int 21h
    ret
PRO4 endp
```

```

PRO5 proc near
    mov dx,offset PROG5
    mov ah,09h
    int 21h
    ret
PRO5 endp

; Procédure pour afficher un saut de ligne
NEWLINE_PROC proc near
    mov dx,offset NEWLINE
    mov ah,09h
    int 21h
    ret
NEWLINE_PROC endp

; Routine d'interruption périodique 1CH
deroute proc near
    push ax
    push bx
    push cx
    push dx

    ; Incrémente le compteur de temps
    inc word ptr [timer_counter]
    cmp word ptr [timer_counter], 91 ; Environ 5 secondes (18.2 ticks/sec * 5 sec)
    jne skip_task

    ; Réinitialise le compteur de temps
    mov word ptr [timer_counter], 0

    ; Passe à la tâche suivante
    inc byte ptr [current_task]
    cmp byte ptr [current_task], 5
    jle call_task
    mov byte ptr [current_task], 1
    call NEWLINE_PROC ; Ajoute un saut de ligne avant de revenir à
la tâche
    call_task:
; Appelle la tâche appropriée selon la valeur de current_task
    cmp byte ptr [current_task], 1
    je execute_task1
    cmp byte ptr [current_task], 2
    je execute_task2
    cmp byte ptr [current_task], 3
    je execute_task3
    cmp byte ptr [current_task], 4
    je execute_task4
    cmp byte ptr [current_task], 5
    je execute_task5
    jmp skip_task

```



```

execute_task1:
    call PRO1
    jmp skip_task

execute_task2:
    call PRO2
    jmp skip_task

execute_task3:
    call PRO3
    jmp skip_task

execute_task4:
    call PRO4
    jmp skip_task

execute_task5:
    call PRO5

skip_task:
    pop dx
    pop cx
    pop bx
    pop ax
    iret
deroute endp

start:
    mov ax, data
    mov ds, ax

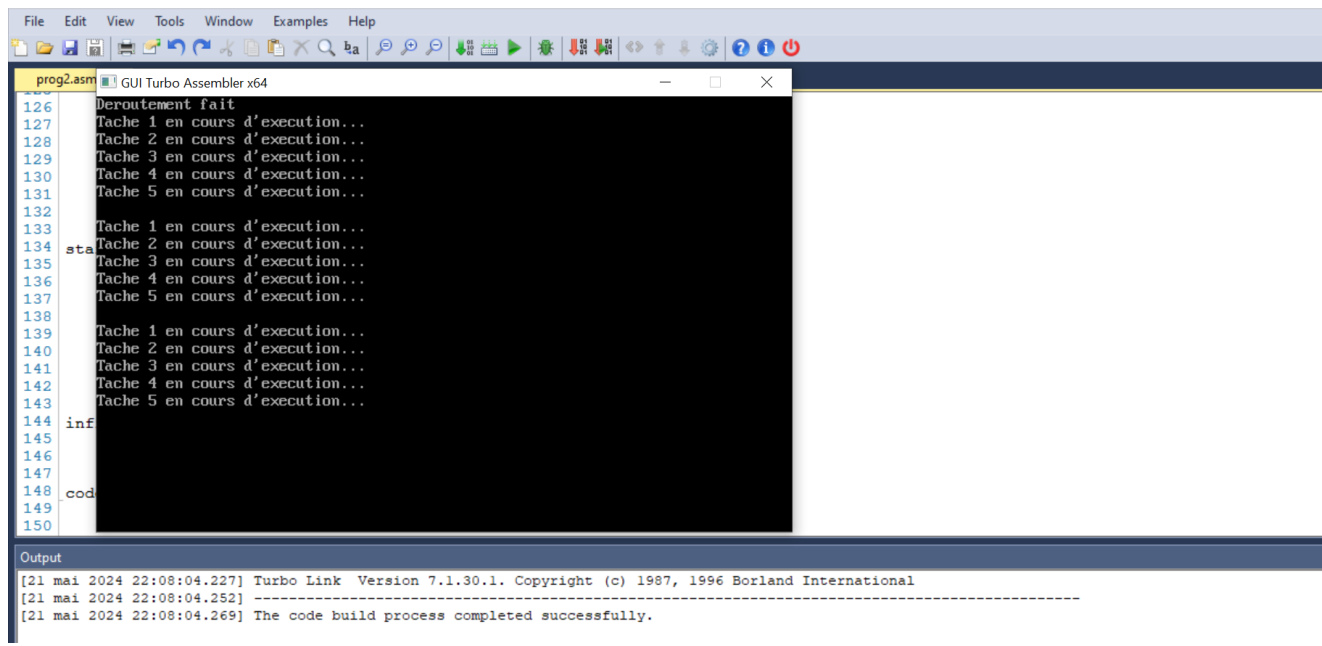
    mov bl, 0H
    mov bh, 1h
    mov ax, 3
    int 10H ; Efface l'écran en utilisant l'interruption 10H
    call instalation

infinie_et_au_dela:
    ; Boucle infinie principale
    jmp infinie_et_au_dela

code ends
end start

```

Capture : il affiche chaque tache après 5 sec et saute la ligne après tache 5



The screenshot shows the GUI Turbo Assembler x64 interface. The main window displays the assembly code for `prog2.asm`. The code is as follows:

```
126 Deroutement fait
127 Tache 1 en cours d'execution...
128 Tache 2 en cours d'execution...
129 Tache 3 en cours d'execution...
130 Tache 4 en cours d'execution...
131 Tache 5 en cours d'execution...
132
133 Tache 1 en cours d'execution...
134 Tache 2 en cours d'execution...
135 sta Tache 3 en cours d'execution...
136 Tache 4 en cours d'execution...
137 Tache 5 en cours d'execution...
138
139 Tache 1 en cours d'execution...
140 Tache 2 en cours d'execution...
141 Tache 3 en cours d'execution...
142 Tache 4 en cours d'execution...
143 Tache 5 en cours d'execution...
144 inf
145
146
147
148 _cod
149
150
```

The output window at the bottom shows the following messages:

```
[21 mai 2024 22:08:04.227] Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International
[21 mai 2024 22:08:04.252] -----
[21 mai 2024 22:08:04.269] The code build process completed successfully.
```

3/ Modifier les deux programmes pour que chacun deux s'arrête après 5 min d'exécution et refaire le travail demandé dans les questions (1) et (2) :

1 minute = 60 secondes

5 minutes = 5 * 60 = 300 secondes

Appels pour 5 minutes = 300 secondes * 18.2 appels/seconde = 5460 appels

Programme2 apres modification :

Modifications apportées :

- 1.Ajout d'un nouveau compteur `total_counter`
- 2.vérification de la durée totale dans la routine d'interruption `deroute`
- 3.Ajout de l'étiquette `end_program` dans la routine d'interruption `deroute`
- 4.Ajout d'une étiquette et d'une séquence de terminaison du programme.

```
data segment
    PROG1 db "Tache 1 en cours
d'execution...",10,13,"$"
    PROG2 db "Tache 2 en cours
d'execution...",10,13,"$"
    PROG3 db "Tache 3 en cours
d'execution...",10,13,"$"
    PROG4 db "Tache 4 en cours
d'execution...",10,13,"$"
    PROG5 db "Tache 5 en cours
d'execution...",10,13,"$"
    NEWLINE db 10,13,"$"
    INSTALLED db "Deroutement
fait",10,13,"$"
    timer_counter dw 0
    current_task db 0
    total_counter dw 0 data ends
```

```
code segment
    assume cs: code, ds: data
```

```
instalation proc near
    push ds
    mov ax,cs
    mov ds,ax
    mov dx,offset deroute
    mov ax,251CH
    int 21H
    pop ds
    mov dx,offset INSTALLED
    mov ah,09h
    int 21h
    ret
instalation endp
PRO1 proc near
    mov dx,offset PROG1
    mov ah,09h
    int 21h
    ret
PRO1 endp
PRO2 proc near
    mov dx,offset PROG2
    mov ah,09h
```

```

    int 21h
    ret
PRO2 endp
PRO3 proc near
    mov dx,offset PROG3
    mov ah,09h
    int 21h
    ret
PRO3 endp
PRO4 proc near
    mov dx,offset PROG4
    mov ah,09h
    int 21h
    ret
PRO4 endp
PRO5 proc near
    mov dx,offset PROG5
    mov ah,09h
    int 21h
    ret
PRO5 endp
NEWLINE_PROC proc near
    mov dx,offset NEWLINE
    mov ah,09h
    int 21h
    ret
NEWLINE_PROC endp

deroute proc near
    push ax
    push bx
    push cx
    push dx
    ; Incrémente le compteur de temps
    inc word ptr [timer_counter]
    inc word ptr [total_counter]
    ; Vérifie si 5 minutes se sont
    écoulées
    cmp word ptr [total_counter], 5460
    jge end_program
    ; Vérifie si 5 secondes se sont
    écoulées
    cmp word ptr [timer_counter], 91 ; Environ 5
    secondes (18.2 ticks/sec * 5 sec)
    jne skip_task
    ; Réinitialise le compteur de temps
    mov word ptr [timer_counter], 0
    ; Passe à la tâche suivante
    inc byte ptr [current_task]
    cmp byte ptr [current_task], 5
    jle call_task
    mov byte ptr [current_task], 1
    call NEWLINE_PROC ; Ajoute un saut de ligne
    avant de revenir à la tâche 1
call_task:
    ; Appelle la tâche appropriée
    cmp byte ptr [current_task], 1
    je execute_task1
    cmp byte ptr [current_task], 2
    je execute_task2
    cmp byte ptr [current_task], 3
    je execute_task3
    cmp byte ptr [current_task], 4
    je execute_task4
    cmp byte ptr [current_task], 5
    je execute_task5
    jmp skip_task

```

```

execute_task1:
    call PRO1
    jmp skip_task

execute_task2:
    call PRO2
    jmp skip_task

execute_task3:
    call PRO3
    jmp skip_task

execute_task4:
    call PRO4
    jmp skip_task

execute_task5:
    call PRO5

skip_task:
    pop dx
    pop cx
    pop bx
    pop ax
    iret

end_program:
    ; Termine le programme
    mov ah, 4Ch
    int 21h
deroute endp

start:
    mov ax, data
    mov ds, ax

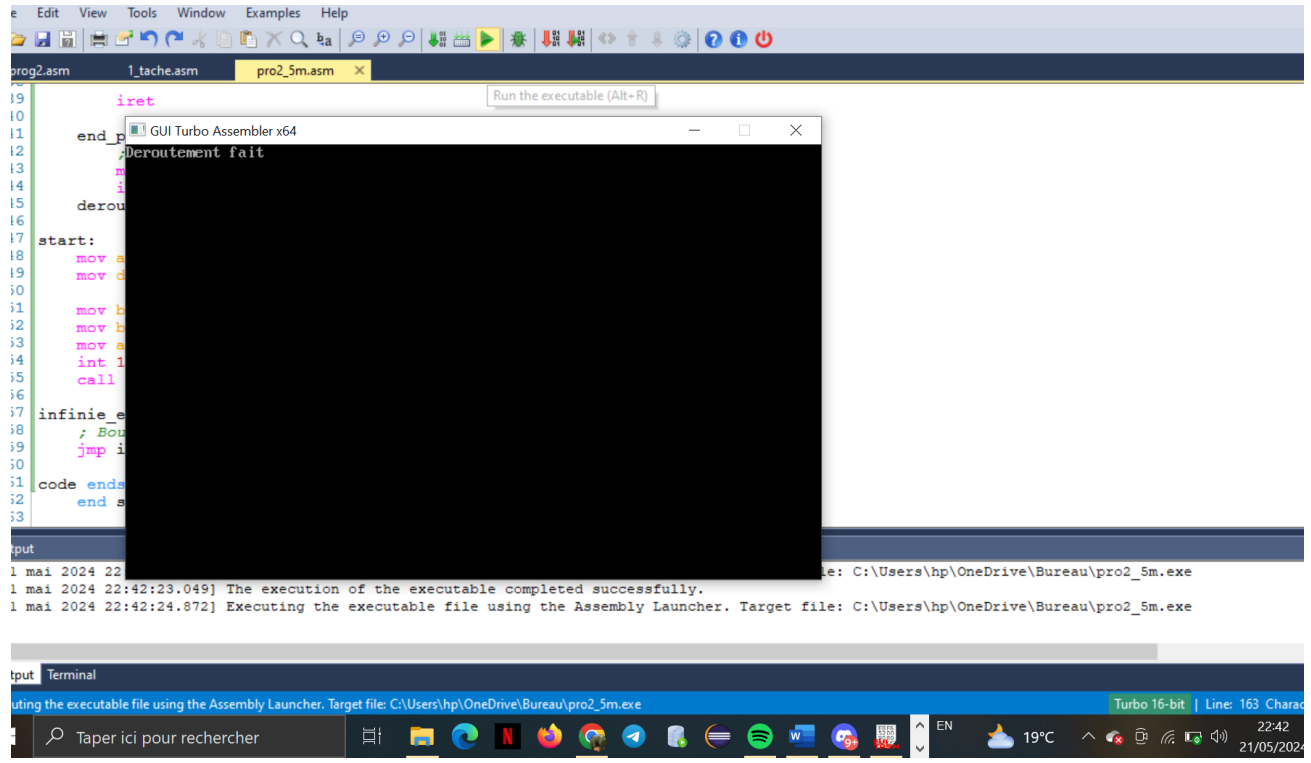
    mov bl, 0H
    mov bh, 1h
    mov ax, 3
    int 10H ; Efface l'écran en
    utilisant l'interruption 10H
    call instalation

infinie_et_au_dela:
    ; Boucle infinie principale
    jmp infinie_et_au_dela

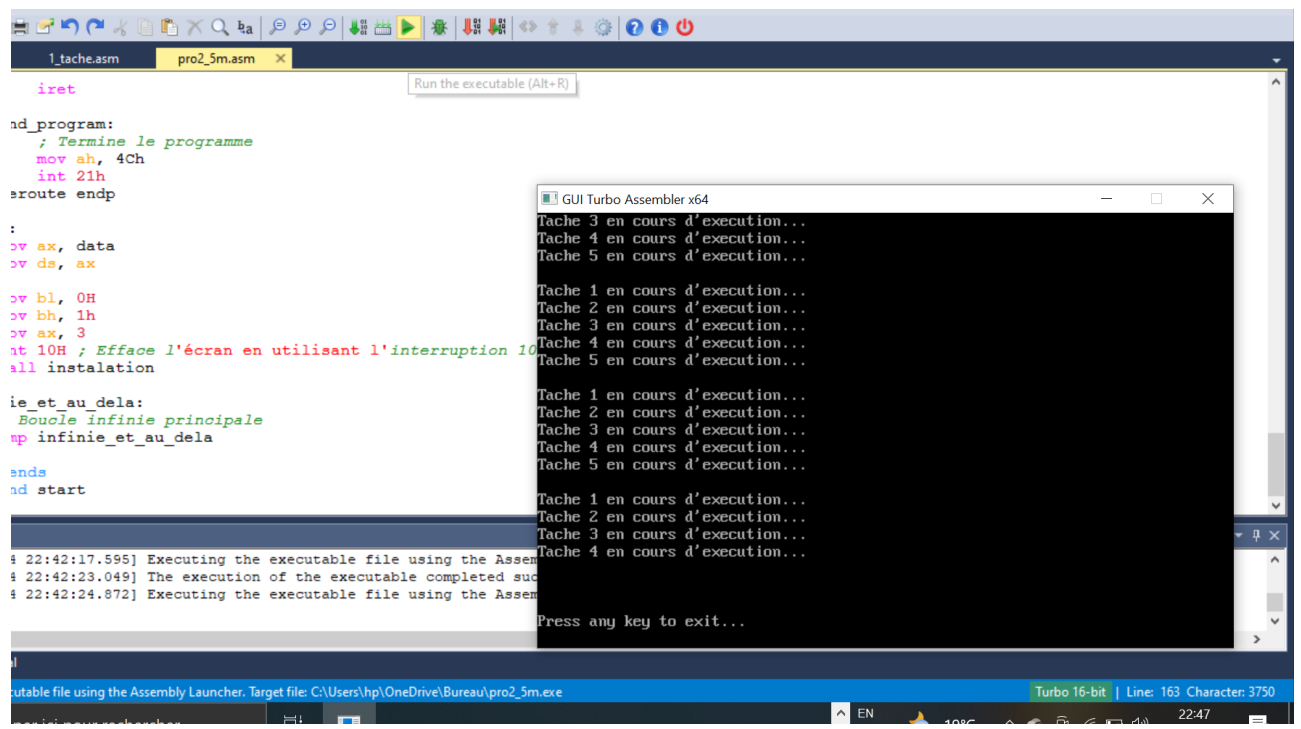
code ends
end start

```

Capture :



Après 5 min :



Programme1 apres modification :

```
data segment
    message db 10,13,"**** Programme
principal en cours **** $" ; Message
principal
    deroute_msg db 10,13,"deroutement
fait... $" ; Message de
deroutement
    debut_msg db 10,13,"****Debut du
quantum de Temps
Logiciel****",13,10,"$" ; Message de
debut de quantum
    sec_msg db "1 sec ecolee..... $"
; Message de décompte de la seconde
data ends
my_stack segment stack 'stack'
    dw 128 dup(?) ; Définition
du segment de pile
    TOP label word ; Pointeur de
sommet de pile
my_stack ends
code segment
    assume cs:code, ds:data,
ss:my_stack ; Assurer les segments de
code, de données et de pile
output proc near
    mov ah, 09h ; Chargement de
la fonction d'affichage du DOS
    int 21h ; Appel de
l'interruption DOS pour afficher la
chaîne
    ret ; Retour de la
procédure
output endp
; Variables pour le comptage du temps
time_counter dw 0 ; Compteur de
temps
deroute_done db 0 ; Indicateur de
deroutement affiché

deroute proc near
    push ax ; Sauvegarde des
registres utilisés
    push dx
    push ds

    ; Si le déroutement a déjà été
affiché, sauter l'affichage
    cmp deroute_done, 1
    je skip_display

    ; Afficher le message de
deroutement
    mov ax, data ; Chargement du
segment de données
    mov ds, ax
    mov dx, offset deroute_msg ;
Chargement de l'adresse du message de
deroutement
    call output ; Appel de la
procédure d'affichage
    ; Marquer le déroutement comme
déjà affiché
    mov deroute_done, 1
    je skip_display
```

```
; Afficher le message de déroutement
mov ax, data ; Chargement dusegment de données
mov ds, ax
mov dx, offset deroute_msg ;
Chargement de l'adresse du message de
deroutement
call output ; Appel de la
procédure d'affichage
; Marquer le déroutement comme
déjà affiché
mov deroute_done, 1

skip_display:
; Incrémenter le compteur de temps
(18,2 fois par seconde)
inc word ptr ds:[time_counter]
cmp word ptr ds:[time_counter],
5400 ; Environ 5 minutes (300 secondes
* 18)
jnb short skip_reset_counter

; Si 5 minutes se sont écoulées,
terminer le programme
mov ah, 4Ch ; Chargement de
la fonction de terminaison du
programme DOS
int 21h ; Appel de
l'interruption DOS pour terminer le
programme

skip_reset_counter:
; Afficher les messages
mov ax, data ; Chargement du
segment de données
mov ds, ax
mov dx, offset debut_msg ;
Chargement de l'adresse du message de
debut de quantum
call output ; Appel de la
procédure d'affichage
mov cx, 3 ; Boucle pour
afficher le message "1 sec écoulée"
trois fois
display_loop:
    mov dx, offset sec_msg ;
Chargement de l'adresse du message de
seconde écoulée
    call output ; Appel de la
procédure d'affichage
    loop display_loop ; Décrémente CX
et saute à "display_loop" tant que CX
n'est pas nul

; Restaurer les registres et
revenir de l'interruption
pop ds ; Restauration
des registres
pop dx
pop ax
iret ; Retour de
l'interruption

deroute endp
```

```

installation proc near
    push ds                ; Sauvegarde des registres utilisés
    mov ax, cs
    mov ds, ax
    mov dx, offset deroute
    mov ax, 251Ch          ; Chargement de la fonction d'installation de l'interruption
    int 21h                ; Appel de l'interruption DOS pour installer l'interruption
1Ch
    pop ds                 ; Restauration des registres
    ret                    ; Retour de la procédure
installation endp

start:
    mov ax, data           ; Chargement du segment de données
    mov ds, ax
    mov ax, my_stack       ; Chargement du segment de pile
    mov ss, ax
    mov sp, TOP            ; Initialisation du pointeur de pile
    call installation; Appel de la procédure d'installation de l'interruption1Ch

    ; Afficher le message principal
    mov dx, offset message
    call output

    ; Boucle d'attente d'une seconde (environ 18,2 ticks par seconde)
    mov cx, 5400           ; 5 minutes en ticks (300 secondes * 18)
wait_loop:
    mov ax, 3d09h          ; Chargement de la fonction d'attente du DOS
    int 21h                ; Appel de l'interruption DOS pour l'attente
    loop wait_loop         ; Décrémente CX et saute à "wait_loop" tant que CX n'est pas
nul

code ends
end start

```

Capture :

The screenshot shows the GUI Turbo Assembler x64 interface. The main window displays the assembly code from the previous block, with line numbers 81 to 105. The code includes comments in French. The output window at the bottom shows the following text:

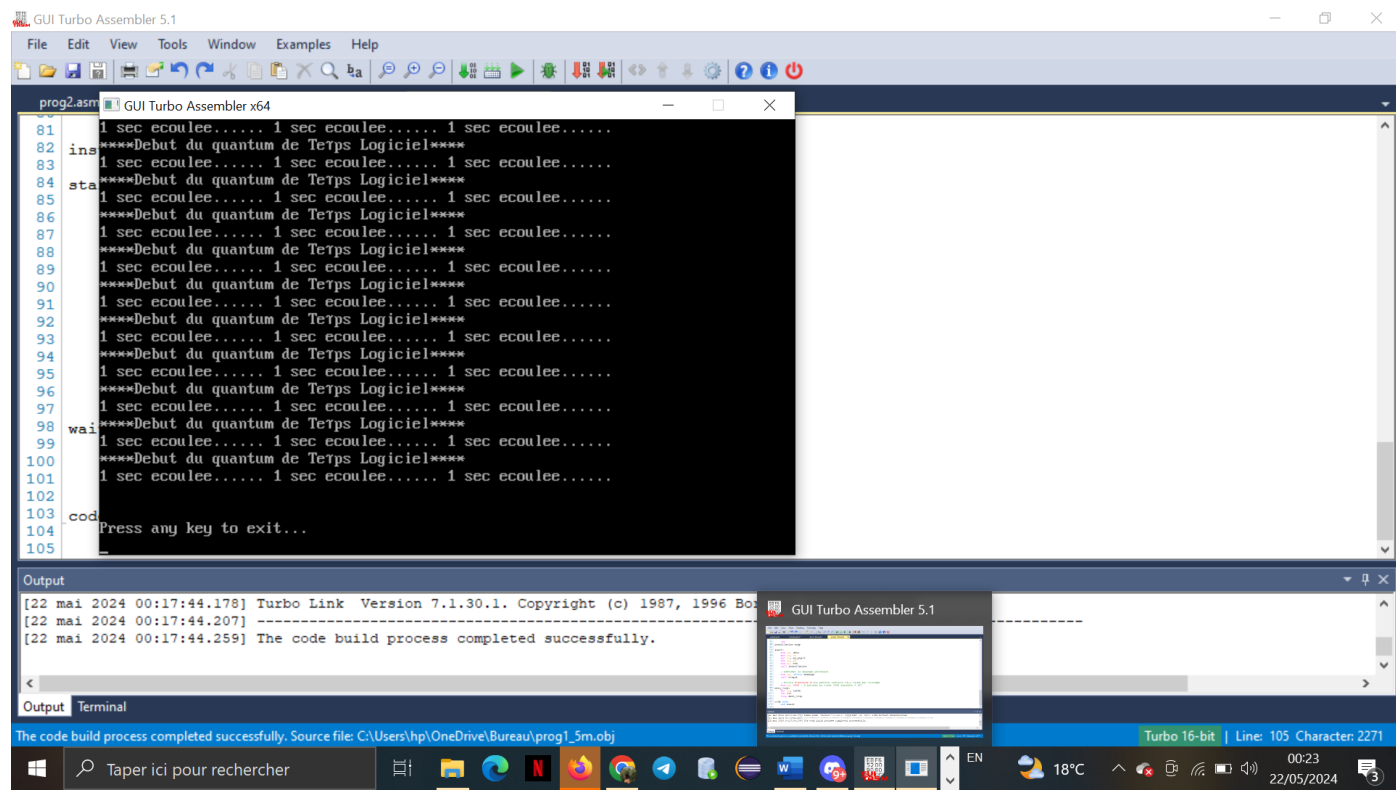
```

[22 mai 2024 00:17:44.178] Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International
[22 mai 2024 00:17:44.207] -----
[22 mai 2024 00:17:44.259] The code build process completed successfully.

```

The status bar at the bottom indicates "The code build process completed successfully. Source file: C:\Users\hp\OneDrive\Bureau\prog1_5m.obj". The taskbar at the bottom shows various application icons and the system clock.

Après 5 min :



The screenshot shows the GUI Turbo Assembler 5.1 interface. The main window displays the assembly code for 'prog2.asm'. The code includes a loop that prints '1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....' and '****Debut du quantum de Temps Logiciel****' 100 times. The output window shows the build process completed successfully. The status bar at the bottom indicates 'Turbo 16-bit | Line: 105 Character: 2271'.

```
81 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
82 ins ****Debut du quantum de Temps Logiciel****
83 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
84 sta ****Debut du quantum de Temps Logiciel****
85 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
86 ****Debut du quantum de Temps Logiciel****
87 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
88 ****Debut du quantum de Temps Logiciel****
89 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
90 ****Debut du quantum de Temps Logiciel****
91 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
92 ****Debut du quantum de Temps Logiciel****
93 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
94 ****Debut du quantum de Temps Logiciel****
95 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
96 ****Debut du quantum de Temps Logiciel****
97 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
98 wai ****Debut du quantum de Temps Logiciel****
99 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
100 ****Debut du quantum de Temps Logiciel****
101 1 sec ecoulee..... 1 sec ecoulee..... 1 sec ecoulee.....
102
103 cod
104 Press any key to exit...
105
```

Output

```
[22 mai 2024 00:17:44.178] Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International, Inc.
[22 mai 2024 00:17:44.207] -----
[22 mai 2024 00:17:44.259] The code build process completed successfully.
```

Output Terminal

The code build process completed successfully. Source file: C:\Users\hpn\OneDrive\Bureau\prog1_5m.obj

Turbo 16-bit | Line: 105 Character: 2271

les ajouts dans le code :

1. Ajout de la variable `time_counter` dw 0` pour compter le temps écoulé.
2. Ajout de la variable `deroute_done` db 0` pour suivre si le message de déroutement a déjà été affiché.
3. Ajout de la procédure `deroute` pour gérer l'affichage des messages et le décompte du temps.
4. Ajout de la logique pour afficher le message de déroutement une seule fois dans la procédure `deroute`.
5. Ajout de la vérification si 5 minutes se sont écoulées dans la procédure `deroute`, et terminer le programme le cas échéant.
6. Modification de la procédure `installation` pour installer l'interruption 1Ch.
7. Modification de la procédure `start` pour appeler la procédure `installation` et afficher le message principal.